ICS1503 WEB APPLICATION DEVELOPMENT

EXPENSE SPLITTER

MINI PROJECT REPORT

Team members:

1. Sandhya Giribabu
   Reg No.: 3122237001045
2. Shinigdapriya Sathish
   Reg No.: 3122237001048
3. Yuvashree P H
   Reg No.: 3122237001062

Problem Statement:

Managing shared expenses among friends, classmates, or colleagues can often become confusing and error-prone, especially when multiple people contribute to different items or activities. The goal of the *Simple Expense Splitter* web application is to simplify this process by allowing users to record expenses, view per-person balances, and track how much each person owes or is owed.

The application provides an intuitive interface to **add, update, and delete expenses**, ensuring each transaction is stored persistently using **MongoDB**. It is built with **Spring Boot (backend)** and **Vue.js (frontend)**, offering **RESTful APIs**, **data validation**, and **real-time UI updates**.

The key features include:

- Add and manage **expenses** (title, amount, payer, and participants).
- Automatically calculate **per-person totals** and balances.
- Perform **CRUD operations** for both *users* and *expenses*.
- **Form validation** and error feedback to prevent invalid submissions.
- **Persistent data storage** using MongoDB with proper schema and relations.
- **Routing** and **local storage** integration in the frontend for better UX.

- **Spring Boot tests** for controller/service and **Vitest** for Vue components.
- **TailwindCSS** used for responsive, accessible UI.
- **JWT authentication** for secure access (bonus).
- **Flyway migrations** for structured database versioning (bonus).

Methodology:

The development of the *Simple Expense Splitter* followed a **modular, full-stack development approach**, ensuring clear separation between frontend, backend, and database layers. The methodology focused on iterative design, implementation, testing, and integration to achieve a reliable and user-friendly application.

1. System Architecture

The project follows a **three-tier architecture**:

- **Frontend (Presentation Layer):** Built using **Vue.js**, responsible for user interaction, input validation, and displaying expense details dynamically.
- **Backend (Application Layer):** Implemented in **Spring Boot**, managing business logic, validations, and communication between frontend and database.

- **Database (Data Layer):** Managed by **MongoDB**, responsible for data persistence, relationships, and query optimization.

Data is exchanged between frontend and backend via **RESTful APIs** using JSON format.

2. Requirements Analysis

Before implementation, requirements were categorized into:

- **Functional Requirements:**
  - Add, edit, view, and delete expenses.
  - Automatically compute per-person totals.
  - Validate user input and prevent incorrect submissions.
  - Support multiple users with secure access.
- **Non-Functional Requirements:**
  - Maintainable and modular codebase.
  - Responsive and accessible UI.
  - Secure authentication and restricted CORS.
  - Persistent data storage using MongoDB.

3. Design Phase

- **Database Design:**
  Two main entities were created — *User* and *Expense* — with a one-to-many relationship.

  - *User* table stores user details such as `id`, `name`, and `email`.
  - *Expense* table stores `expense_id`, `title`, `amount`, `paid_by`, and list of participants.
  - **Flyway migrations** were implemented for maintaining schema changes systematically.

- **API Design:**
  RESTful APIs were defined using standard routes such as:

  - `GET /expenses` — Retrieve all expenses
  - `POST /expenses` — Add new expense
  - `PUT /expenses/{id}` — Update expense
  - `DELETE /expenses/{id}` — Delete expense
    Each API returns appropriate HTTP status codes (200, 201, 400, 404).

- **UI Design:**
  The user interface was designed using **Vue components** and styled

with **TailwindCSS**.

- Routes were configured with **Vue Router** (e.g., `/home`, `/add-expense`, `/summary`).
- Browser storage (local/session) was used to preserve user session and temporary form data.

4. Implementation Phase

- **Backend Implementation (Spring Boot):**
  - Developed controllers, services, and repositories following layered architecture.
  - Implemented **input validation** using annotations (`@Valid`, `@NotNull`, etc.).
  - Configured **JWT authentication** and **CORS restrictions** for secure access.
  - Used **Spring Data JPA** for ORM mapping with MongoDB.
- **Frontend Implementation (Vue.js):**
  - Developed reusable components for forms, expense lists, and summary views.

- Implemented **reactive data binding** and form validation using Vue directives.
- Added **error messages and disabled buttons** for invalid inputs.
- Integrated backend APIs using **Axios** for HTTP requests.

5. Testing and Validation

Testing was performed at multiple levels:

- **Backend Testing:**
  - Unit tests written using **Spring Boot Test** for controller and service layers.
  - Verified CRUD functionality and calculation logic.
- **Frontend Testing:**
  - Component tests written using **Vitest** to ensure form inputs and computed values worked correctly.
- **Integration Testing:**
  - Checked complete flow from frontend submission to backend storage and UI update.

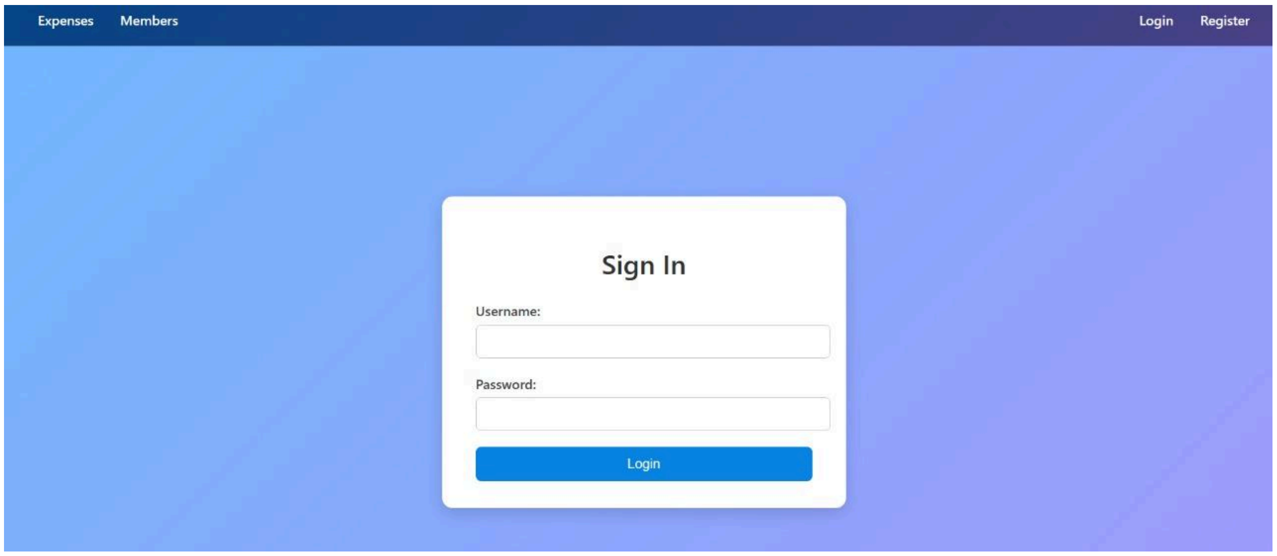
6. Deployment and Version Control

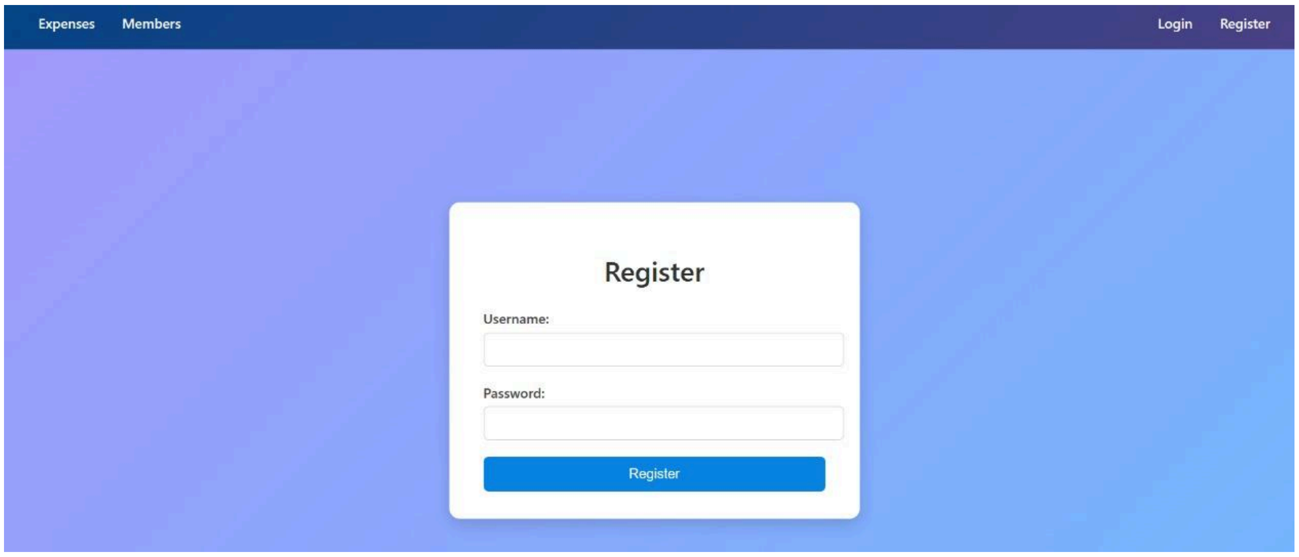
- The entire project was version-controlled using **GitHub**, maintaining clear commit history and documentation.
- Database migrations handled by **Flyway** ensured smooth deployment and schema consistency across environments.
- The application was tested on a local development server with the ability to later deploy on cloud or institutional infrastructure.

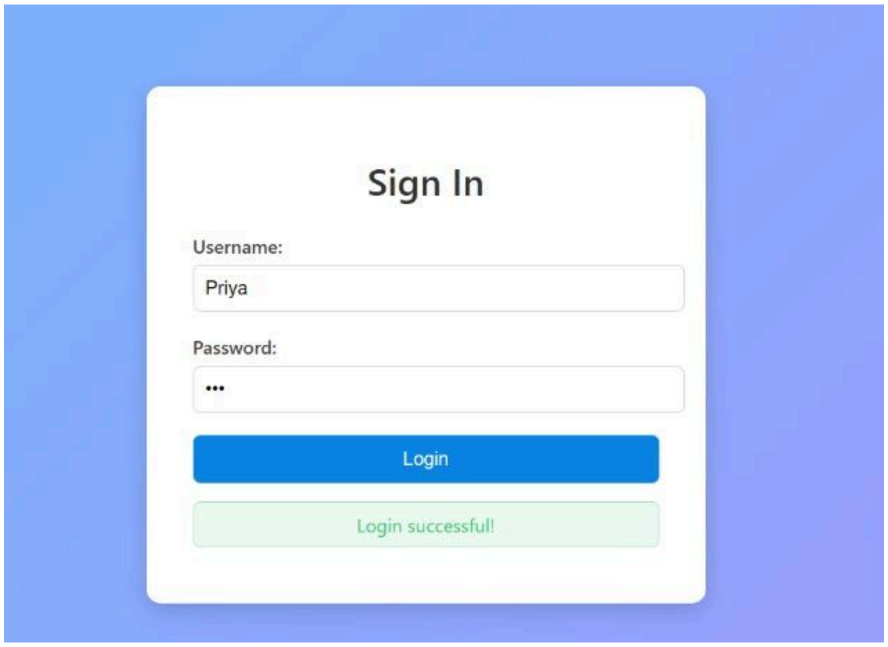
7. Tools and Technologies Used

Layer	Technology	Purpose
Frontend	Vue.js, TailwindCSS, Axios, Vitest	UI development, styling, API calls, testing
Backend	Spring Boot, Spring Data JPA, Flyway, JWT	Business logic, ORM, migrations, security
Database	MongoDB	Persistent data storage
Version Control	Git, GitHub	Collaboration and source management
Build Tools	Maven, npm	Dependency management

Output


Expenses
Members
Login
Register
Sign In
Username:
Password:
Login


Expenses
Members
Login
Register
Register
Username:
Password:
Register

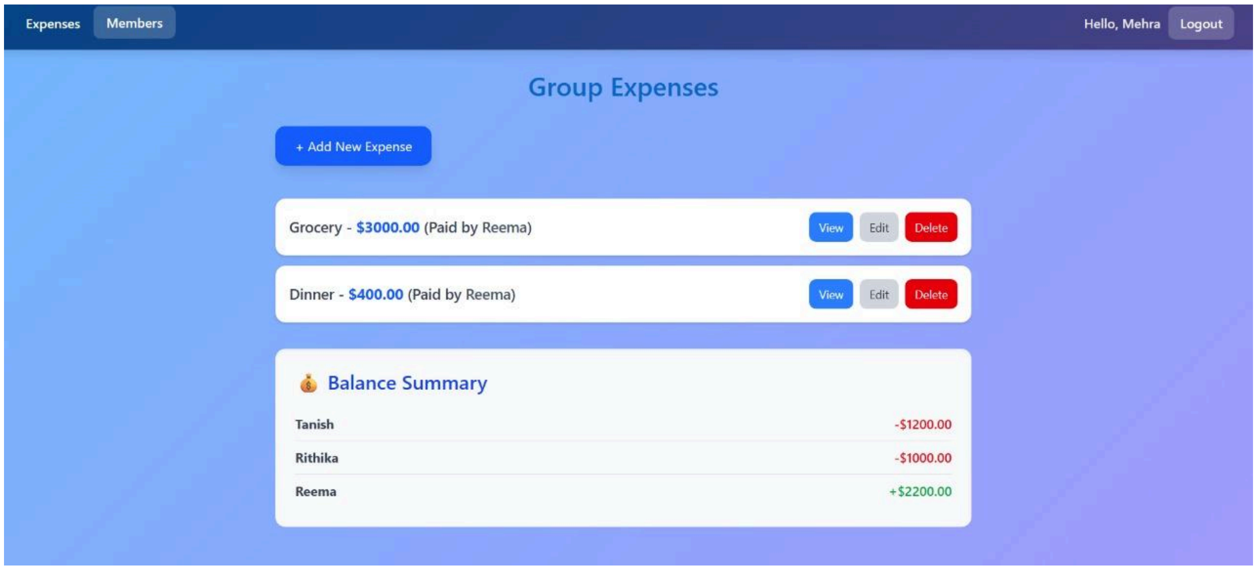


Sign In

Username: Priya

Password: •••

Login successful!



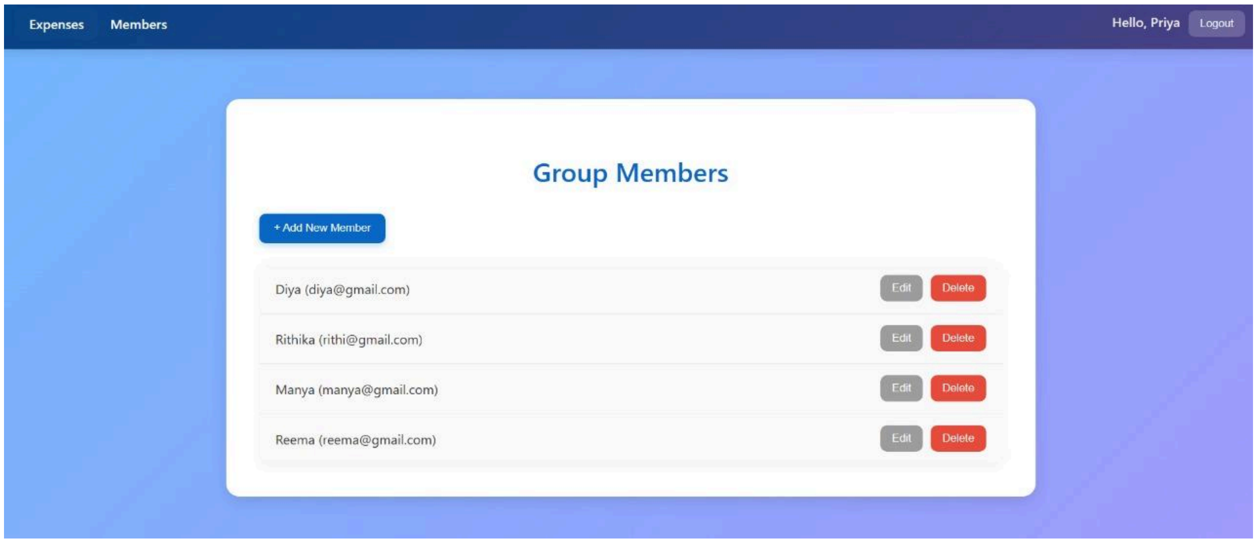
Group Expenses

Grocery - $3000.00 (Paid by Reema)

Dinner - $400.00 (Paid by Reema)

💰 Balance Summary

Tanish	-$1200.00
Rithika	-$1000.00
Reema	+$2200.00



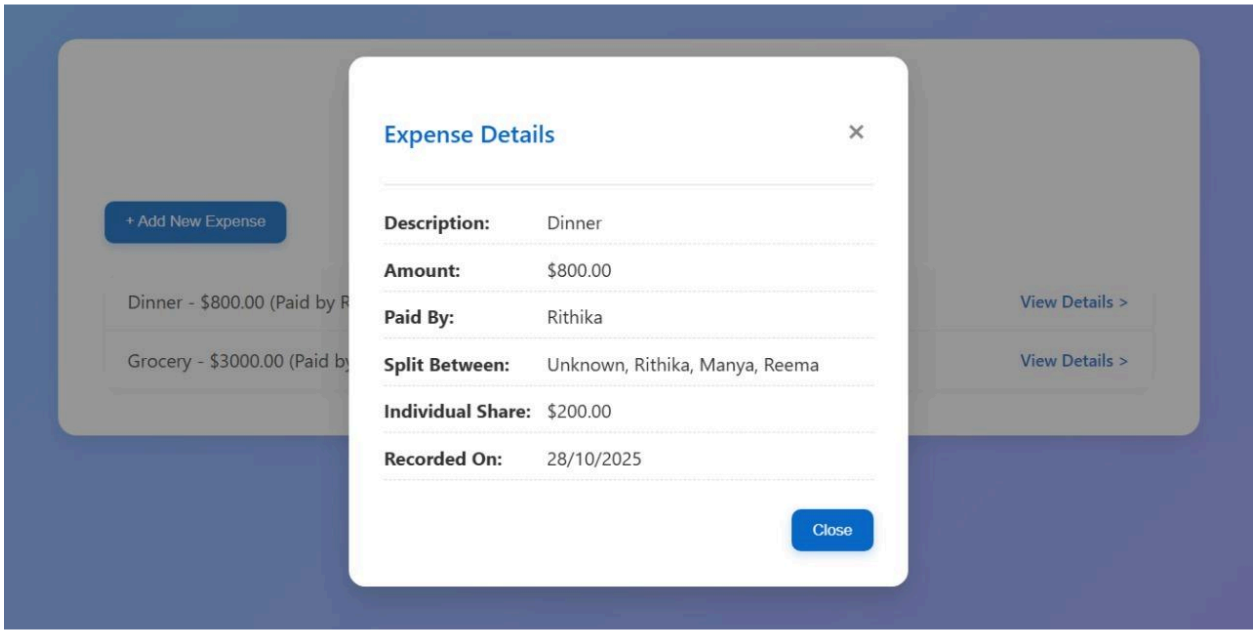
Group Members

Diya (diya@gmail.com)

Rithika (rithi@gmail.com)

Manya (manya@gmail.com)

Reema (reema@gmail.com)

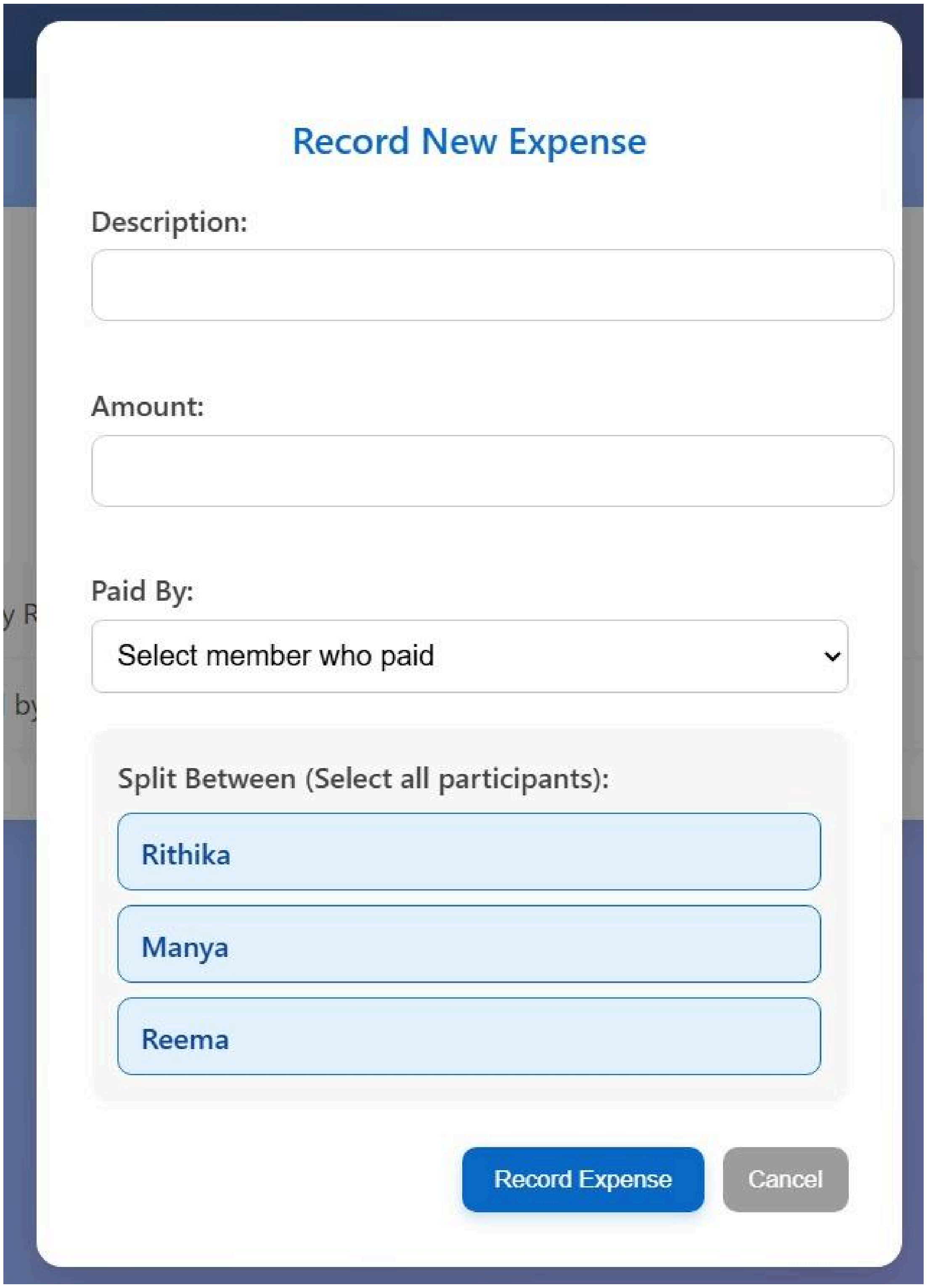


Dinner - $800.00 (Paid by R

Grocery - $3000.00 (Paid by

Expense Details

Description:	Dinner
Amount:	$800.00
Paid By:	Rithika
Split Between:	Unknown, Rithika, Manya, Reema
Individual Share:	$200.00
Recorded On:	28/10/2025



Record New Expense

Description:

Amount:

Paid By:

Select member who paid

Split Between (Select all participants):

Rithika

Manya

Reema

GitHub Repository Link

https://github.com/yuvashreeph/SimpleExpenseSplitter

Results and Conclusion

The *Simple Expense Splitter* was successfully implemented as a full-stack web application that met all the functional and non-functional requirements.

Functional Results:

- Two classes — *User* and *Expense* — were implemented with complete **CRUD functionality**.
- The application demonstrated **end-to-end working flow**, from frontend input forms to backend API persistence and database storage.
- **RESTful APIs** were designed following standard conventions (`GET`, `POST`, `PUT`, `DELETE`) with clear JSON responses and proper HTTP status codes.
- **MongoDB integration** ensured stable, persistent, and relational data storage.
- The **frontend**, built using Vue with **Vue Router**, provided smooth navigation and inline form validation.
- **Error handling** was implemented at both frontend and backend levels, displaying meaningful messages to users.

- **Unit tests** validated key components and services, confirming the correctness of calculations and API responses.
- The use of **TailwindCSS** improved UI accessibility and responsiveness, while **JWT authentication** and **CORS restrictions** enhanced security.
- **Flyway migrations** ensured reliable and trackable database updates during development.

Conclusion:

The project achieved its intended goal of simplifying group expense management through a clean, secure, and user-friendly web application. The system can be extended in the future to include advanced features like group settlements, currency conversion, or integration with payment gateways.

Learning Outcomes

Developing the *Simple Expense Splitter* project provided hands-on experience across the full web development stack. Key takeaways include:

1. **Backend Development:**
    - Designing RESTful APIs with **Spring Boot**.
    - Implementing **CRUD operations** and **data validation** using Spring annotations.
    - Handling exceptions and returning user-friendly error messages.

2. **Database Management:**

    - Structuring relational data in **MongoDB** with foreign key relationships.

    - Applying **Flyway migrations** for schema version control.

3. **Frontend Development:**

    - Building responsive and modular UIs using **Vue.js**.

    - Implementing **routing**, **state management**, and **form validation**.

    - Using **TailwindCSS** for a clean and accessible design.

4. **Integration & Testing:**

    - Connecting frontend and backend through **REST APIs**.

    - Writing **Spring Boot tests** (controller/service) and **Vue Vitest** component tests.

    - Managing **CORS**, **authentication**, and **data persistence** effectively.

5. **Project Management & Collaboration:**

    - Maintaining a clean directory structure with meaningful code documentation.

- ○ Using **GitHub** for version control, commits, and project presentation.

Overall, the project reinforced practical knowledge in **full-stack application development**, **testing**, and **secure deployment practices**, resulting in a robust, functional, and user-friendly web solution.